Relatório do Projeto: Ordenação por Inserção Binária

Brenda Barbosa de Souza (202037622)
Thelma Evangelista dos Santos (231003513)

Dezembro 2025

1 O Processo

1.1 Instalações

A primeira etapa do projeto consistiu em configurar o ambiente de desenvolvimento do Coq. Como a versão Web do Coq não estava disponível no momento, tentamos utilizá-lo localmente tanto pelo terminal quanto pelo VSCode.

A instalação do Coq em si foi relativamente simples, e o comando `coqtop` funcionou normalmente no terminal. Entretanto, a integração com o VSCode apresentou vários problemas. A cada tentativa de carregar um arquivo `.v`, mensagens como *"coqtop is not running"* e *"command not found"* apareciam. Além disso, as extensões disponíveis, VSCoq Legacy e VSROCQ, não funcionavam de maneira consistente.

Foi necessário reinstalar extensões, testar diferentes versões e reiniciar o VSCode diversas vezes. Depois de tentativas repetidas, conseguimos estabilizar a extensão VSCoq e finalmente iniciar o desenvolvimento do projeto.

1.2 Escrevendo o algoritmo

Antes de iniciarmos a formalização baseada nas especificações do professor, já possuíamos uma versão funcional, feita na disciplica Lógica Computacional 1, de um algoritmo de ordenação por inserção binária. Essa versão utilizava uma outra definição de lista ordenada, implementada manualmente. Embora estivesse correta e bem estruturada, ela não utilizava a biblioteca padrão `Sorted` do Coq, que era explicitamente exigida no enunciado do projeto.

Diante desse cenário, tínhamos duas opções:

1. Reescrever todo o código para se alinhar exatamente à base proposta pelo professor;

2. Manter as estruturas e provas já concluídas, adicionando somente o necessário para compatibilizar nossa abordagem com a biblioteca exigida.

Decidimos pela segunda opção, pois garantiu menor retrabalho e preservou boa parte do desenvolvimento já realizado.

Diferenças estruturais entre a nossa base e a base do professor

A função `binsert` fornecida pelo professor utiliza um `match` explícito para tratar o caso em que a lista `l2` pode ser vazia. Já a nossa implementação utilizava `hd 99 l2` para acessar o primeiro elemento da segunda metade, com um valor padrão. Essa diferença estruturou várias partes das nossas provas, tornando inviável substituir diretamente a nossa versão sem comprometer todo o trabalho já feito.

Por esse motivo, optamos por manter nossa estrutura e apenas garantir que:

- a função `binsert` produz uma lista ordenada;

- a função preserva os elementos originais (i.e., gera uma permutação);

- a definição de ordenação utilizada se relaciona corretamente com a definição oficial `Sorted`.

Adaptação final

Para tentar cumprir a especificação, incluímos ao final do arquivo um lema que estabelece que, se uma lista é ordenada de acordo com nossa definição auxiliar, então ela também é ordenada segundo a definição da biblioteca `Sorted le`. Esse lema funciona como uma espécie de "tradutor" entre as duas abordagens e permite que o teorema final tenha o tipo exigido pelo professor.

2 Aprendizados

- O Coq é extremamente rigoroso pois pequenas diferenças estruturais em funções podem inviabilizar provas inteiras.

- A divisão de provas em lemas menores é essencial para manter clareza e evitar retrabalho.

- Em alguns casos, adaptar uma abordagem já funcional é mais eficiente do que recomeçar uma implementação inteira do zero.

- A formalização de algoritmos comuns em Coq traz uma compreensão mais profunda sobre sua corretude, ordenação e propriedades de permutação.